Machine Learning Engineer Nanodegree

Capstone Proposal

Charlie Garavaglia
January 12, 2018

Proposal

Domain Background

This project is taken from a Kaggle competition, the Statoil/C-CORE Iceberg Classifier Challenge, ongoing as of the above date.

From the description:

> Drifting icebergs present threats to navigation and activities in areas such as offshore of the East Coast of Canada.
>
> Currently, many institutions and companies use aerial reconnaissance and shore-based support to monitor environmental conditions and assess risks from icebergs. However, in remote areas with particularly harsh weather, these methods are not feasible, and the only viable monitoring option is via satellite.
>
> Statoil, an international energy company operating worldwide, has worked closely with companies like C-CORE. C-CORE have been using satellite data for over 30 years and have built a computer vision based surveillance system. To keep operations safe and efficient, Statoil is interested in getting a fresh new perspective on how to use machine learning to more accurately detect and discriminate against threatening icebergs as early as possible.
>
> In this competition, you're challenged to build an algorithm that automatically identifies if a remotely sensed target is a ship or iceberg. Improvements made will help drive the costs down for maintaining safe working conditions.

Problem Statement

As said, the aim of the competition is to distinguish an iceberg from a ship in satellite radar data, specifically images from the Sentinel-1 satellite constellation's C-Band radar. Each image provided contains only an iceberg or an ocean-going ship. Thus, labeling each entry in the test set as 0 for ship or 1 for iceberg is perfectly reasonable, as is producing a probability that the image contains an iceberg. Thus the problem is quantifiable. Indeed, each submission is measured using the log loss between our predicated probability values (of presence of an iceberg) versus ground truth. For difficult classification cases we cannot clearly observe the ground truth, but must rely on provided verification; but in any case, the problem is measurable. Finally, this problem is replicable so long as there are icebergs and ships floating in the sea and satellites to take pictures of them.

A detailed background can be found on the Kaggle Background page.

Datasets and Inputs

Training and testing data are provided in json format. Each entry in train.json and test.json represents an image composed of radar data. The satellite transmits a radar pulse and then records the echo of that radar bouncing off objects. This energy reflected back to the satellite is referred to as *backscatter*. Additionally, this particular satellite is side looking radar, so it records radar information at an angle, rather than objects directly below itself. This angle is called the *incidence angle*, and the background page linked above has the interesting note that "generally, the ocean background will be darker at a higher incidence angle". Finally, Sentinel-1 can transmit and receive radar energy in different planes. This is called the *polarization* of the radar band.

An entry in train.json and test.json contains the following fields:

- **id** = ID of the image
- **band_1, band_2** = list of flattened image data in two bands. Each band is a 75x75 pixel image, so each list has 5625 elements. Values are dB (decibels) of radar backscatter at a given incidence angle and polarization. The polarization of band_1 is HH, where the radar is transmitted and received in the horizontal plane. The polarization of band_2 is HV, i.e. transmit horizontally and receive vertically.
- **inc_angle** = incidence angle of radar image. Some fields of this are marked as na
- **is_iceberg** = target variable; 1 if iceberg, 0 if ship. This exists only in train.json

Solution Statement

Employing computer vision to detect if an object is in an image should solve the problem. This can be accomplished by building a convolutional neural network and then training it on many images containing (or not containing) the desired object, in our case the iceberg. Once trained, a probability that the image contains an iceberg can be outputted, thus the solution is quantifiable, measurable, and since the neural net can be used on many images, it is also replicable.

Benchmark Model

I will endeavor to employ a keras CNN for image classification to recognize icebergs in the radar data. Much like the dog project, I will employ successive convolutional and then max pooling layers to deepen the spatial information while decreasing the spatial dimension of each image. Further, dropout and batch normalization layers will be employed to combat overfitting, before sending it to multiple layers of fully-connected nodes to illuminate any patterns in the data. Finally, a single node with a sigmoid activation will output the probability that the image "contains iceberg". This probability will be compared to the true class label for every prediction, and the log loss will be reported.

Unspecified preprocessing will also occur, but I haven't figured out how I will scale the radar data. To also be considered is the influence of the incidence angle. Perhaps calibration based on the ocean reflectance could be employed, since as the background states "the ocean background will be darker at a higher incidence angle". I will also expand the training set using ImageDataGenerator.

In this section, provide the details for a benchmark model or result that relates to the domain, problem statement, and intended solution. Ideally, the benchmark model or result contextualizes existing methods or known information in the domain and problem given, which could then be objectively compared to the solution. Describe how the benchmark model or result is measurable (can be measured by some metric and clearly observed) with thorough detail.

Evaluation Metrics

As mentioned, the official evaluation metric is the logarithmic loss, or "log loss", of our predictions. If our prediction matches the ground truth, a confident prediction probability will contribute little to the total log loss. However, if our prediction does not match, a confident incorrect prediction will contributed heavily to the log loss. We shall endeavor to minimize the log loss; a perfect classifier (that outputs 1.0 for all iceberg images and 0.0 for all non-iceberg images) would have a log loss equal to zero.

Such a metric is appropriate as it distinguish performance from models who output the same classification scheme for the same dataset. When combined with an accuracy metric for private use, we can derive some insight into how our model is performing.

Detailed explanations of log loss are available online, but here is its formula for a binary classifier:

$$-\frac{1}{N} \sum\limits_{i=1}^N {y_i \log p_i + (1 - y_i) \log (1 - p_i)}$$

were y_i is the prediction for image i (1 = iceberg, 0 = no iceberg), p_i is the probability that image i contains an iceberg, and N is the total number of images.

Note that for each image, only one side of the sum adds to log loss.

Project Design

Data Setup

The train.json file will need to be loaded into python and unpacked. Then the features will need to be separated from the labels, the former into a X_train vector and the later to a y_train one. Similar preparations will be made to the test.json file to obtain X_test and y_test vectors.

In preparation for fitting the model, I also will divide the training data into training and validation subsets, investigating if k-fold validation will be helpful during the process.

Preprocessing Images

This section of the project will require the most research and planning. Our first step will be to visualize the two bands of the image for a small subset of the training data by simple plots. This will give us an idea of what we will be trying to classify.

Next, analysis must be conducted on how to relate ocean brightness with incidence angle. Perhaps once this is known, the ocean background radar values can be standardized, allowing the CNN to only learn the shape of the object in question. At this time, however, I do not know how to best leverage the incidence angle field information. [An idea, possibly crazy, that occurred to me just before this submission is that a constant third channel could be constructed from the incidence angle. While the value would not be a decibel level like each pixel in the other channels, the model might still be able to find connections and make predictions based on it.]

Another avenue is adding an additional channel that is the average of the first two bands.

Additionally, the two bands of the radar image must be reshaped into 75x75 pixel images, and then packaged into tensors for consumption by a keras model.

Finally, the data from each picture will potentially be scaled or normalized by centering the mean decibel value about zero with standard deviation 1, whatever is appropriate (possibly neither). Furthermore, data augmentation will be employed to enlarge the training set, thereby hopefully gaining more predictive power. Both will be accomplished by using the ImageDataGenerator in keras.

Gather Baseline Statistics

We know a perfect classifier would have a log loss of exactly zero, but we don't know how an unintelligent classifier would fare. I would take the step of obtaining the log losses for 3 baseline classifiers: one that always classifies an image as containing an iceberg at probability 1.0, one that always outputs iceberg probability 0.5, and one that always outputs probability 0.0.

This gives us a baseline to see how much improvement our model has made.

Model Construction

The CNN architecture will follow suggested designs and consist of 3 successive Conv2D and MaxPooling2D layers, with overfit-fighting Dropout layers sprinkled in. I am partial to a convolutional window of at least 3x3 or 4x4, and a pooling size of at least 2 or 3, but will bow to the best performance after successive runs. All activations on these layers will be relu following current best practices. (I am also considering adding BatchNormalization layers as the image data will pass through many layers and filters.) The amount of filters will be higher than the dog project, as I have a desktop with a GPU capable of integrating with keras and TensorFlow. Hopefully this will lead to better predictive power.

After the spatial dimensions are shrunk and the spatial information deepened, the image will be transformed into a vector by using a Flatten layer, and then passed to at least 2 layers of Dense fully connected layers also with relu activations. These layers of nodes will serve to elucidate any hidden patterns in the data.

A final Dense node with a sigmoid activation will output the desired probability.

The number of training epochs will be arbitrary, as training will be stopped using an EarlyStopping checkpoint to determinate when validation loss has stopped improving, and the best model will be saved using ModelCheckpoint.

Model Fit and Testing

Once the CNN is compiled using binary_crossentropy (aka log loss) as the loss function, an appropriate optimizer such as rmsprop or adam (further investigation needed), and accuracy as the metric for convenience, we can then use it for training.

Since we've created an EarlyStopping and ModelCheckpoint callback, as well as a ImageDataGenerator, we can pass in the appropriate values to .fit_generator. We validate our model by splitting the training data into training and validation sets, and monitor the validation log loss to see when we can stop. Further investigations will be made to see how k-fold validation might help in this project.

All that's left is to load the model with the best validation loss, and then call model.evaluate on the test images. Once there we can package our submission as Kaggle requires, but this is outside the scope of the capstone proposal.